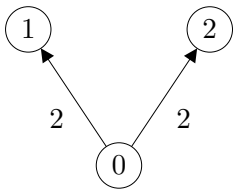
TP 3 - Option informatique - MP* - CCS 2020 - Correction

Question 1 –

```
(* Renvoie 1 si i est avant j dans b et -1 sinon
   Hypothèse: i <> j *)
let rec duel_bul (i: candidat) (j: candidat): bulletin -> int = function
  | [] -> failwith "duel_bul: i et j ne sont pas dans le bulletin"
  | k :: q when k = i -> 1
  | k :: q when k = j -> -1
  | _ :: q -> duel_bul i j q;;
```

```
(* Je ne sais pas si on peut supposer i <> j, dans le doute, je
   renvoie 0 dans ce cas *)
let rec duel (i: candidat) (j: candidat): urne -> int = function
  | _ when i = j -> 0
  | [] -> 0
  | b :: q -> (duel_bul i j b) + (duel i j q);;
```

Question 2 – Voici le graphe de préférence et la matrice d'adjacence :



$$\begin{pmatrix} 0 & 2 & 2 \\ -2 & 0 & 0 \\ -2 & 0 & 0 \end{pmatrix}$$

Question 3 –

⋆ La matrice est antisymétrique car la valeur obtenue lors de la comparaison des candidats i et j est l'opposé de la valeur obtenue lors de la comparaison des candidats j et i.

⋆ Soient $i \neq j$ deux numéros de candidats, soit p_1 le nombre de votants qui placent i avant j et p_2 le nombre de votants qui placent j avant i. Alors $p_1 + p_2 = p$ et la valeur obtenue lors de la comparaison de i et j est $v = p_1 - p_2$. En réduisant modulo 2, on obtient :

$$\begin{aligned} v &\equiv p_1 - p_2 \mod 2 \\ &\equiv p_1 + p_2 \mod 2 \\ &\equiv p \mod 2 \end{aligned}$$

Ainsi, tous les coefficients non diagonaux de la matrice d'adjacence ont la même parité (qui est la parité du nombre de votants).

Question 4 –

```
let depouillement (n: int) (u: urne): int array array =
  let mat_P = Array.make_matrix n n 0 in
  for i = 0 to n-1 do
    for j = 0 to i-1 do
      let v = duel i j u in
      mat_P.(i).(j) <- v;
      mat_P.(j).(i) <- -v;
    done;
  done;
  mat_P;;
```

Question 5 – Il faut que :

- Les deux votants aient placé i avant j.
- Pour tout k, ℓ, si $\{k,\ell\} \neq \{i,j\}$ alors les deux votants n'ont pas placé k et ℓ dans le même ordre.

On note `li1` (resp. `li2`) la liste contenant tous les éléments de $[\![0;n-1]\!] \setminus \{i,j\}$ triée par ordre croissant (resp. décroissant). Une urne possible est :

```
[[i;j] @ li1; li2 @ [i;j]]
```

Montrons que cette urne convient. Soient (k,ℓ) deux numéros de candidats avec $k \neq \ell$. Alors :

- Si $k = i$ et $\ell = j$ alors la valeur obtenue lors de la comparaison de k et ℓ est 2.
- Si $k = j$ et $\ell = i$ alors la valeur obtenue lors de la comparaison de k et ℓ est -2.
- Si $k \in \{i,j\}$ et $\ell \notin \{i,j\}$ alors la valeur obtenue lors de la comparaison de k et ℓ est $1-1=0$.
- Si $k \notin \{i,j\}$ et $\ell \in \{i,j\}$ alors la valeur obtenue lors de la comparaison de k et ℓ est $-1+1=0$.
- Sinon, $k \notin \{i,j\}$ et $\ell \notin \{i,j\}$ alors la valeur obtenue lors de la comparaison de k et ℓ est 0.

Question 6 –

Remarque. Il y a un point que je ne suis pas sûr de comprendre : quel est le type des objets U_1 et U_2 ? Si ce sont des ensembles, alors deux votes ne peuvent pas être identiques, ce qui semble étrange. Si ce sont des listes, alors il s'agit d'une concaténation plutôt que d'une union. Dans la suite, je suppose que ce sont des listes et que l'union est une concaténation.

$$\text{On a } \boxed{M_3 = M_1 + M_2}.$$

En effet, soit n un nombre de candidats et i et j deux entiers naturels strictement inférieurs à n. On note :

- m_1, m_2 et m_3 les coefficients d'indices (i,j) de M_1, M_2 et M_3.
- a_1 et a_2 le nombre de votants qui placent i avant j dans U_1 et U_2.
- b_1 et b_2 le nombre de votants qui placent j avant i dans U_1 et U_2.

Alors :

$$m_1 = a_1 - b_1 \qquad\qquad m_2 = a_2 - b_2$$

De plus :

- Le nombre de votants qui placent i avant j dans U est $a_1 + a_2$.
- Le nombre de votants qui placent j avant i dans U est $b_1 + b_2$.

Donc :

$$m_3 = a_1 + a_2 - b_1 - b_2 = m_1 + m_2$$

Question 7 – Soit $M = (m_{i,j})_{(i,j)\in[\![0;n-1]\!]^2}$ une matrice antisymétrique à coefficients pairs alors :

$$M = \sum_{(i,j)\in I} \frac{m_{i,j}}{2} E_{i,j,n} \qquad \text{où} \qquad I = \Big\{(i,j) \in [\![0;n-1]\!]^2 : m_{i,j} > 0\Big\}$$

Avec les questions précédentes, on obtient :

$$\begin{aligned} M &= \sum_{(i,j)\in I} \frac{m_{i,j}}{2} E_{i,j,n} \\ &= \sum_{(i,j)\in I} \sum_{k=1}^{\frac{m_{i,j}}{2}} E_{i,j,n} \\ &= \sum_{(i,j)\in I} \sum_{k=1}^{\frac{m_{i,j}}{2}} Mat(U_{i,j,n}) \\ &= Mat\left(\bigcup_{(i,j)\in I} \bigcup_{k=1}^{\frac{m_{i,j}}{2}} U_{i,j,n} \right) \end{aligned}$$

D'où le théorème de McGarvey.
Notons que cette égalité peut aussi s'écrire :

$$M = Mat\left(\bigcup_{i=0}^{n-1} \bigcup_{j=0}^{n-1} \bigcup_{k=1}^{\frac{m_{i,j}}{2}} U_{i,j,n} \right),$$

en effet, si $(i,j) \notin I$ alors l'union sur k est vide.

Question 8 – Les deux fonctions intermédiaires construisent les deux bulletins donnés à la question 5.

```
let bul1 (i: int) (j: int) (n: int): int list =
  let rec aux k = match k with
    | k when k = i || k = j -> aux (k+1)
    | k when k = n -> []
    | k -> k :: aux (k+1)
  in
  i :: j :: aux 0;;
```

```
let bul2 (i: int) (j: int) (n: int): int list =
  let rec aux k = match k with
  | k when k = i || k = j -> aux (k-1)
  | k when k = -1 -> i :: j :: []
  | k -> k :: aux (k-1)
  in
  aux (n-1);;
```

```
let mcgarvey (mat_P: int array array): urne =
  let n = Array.length mat_P in
  let u = ref [] in
  for i = 0 to n-1 do
    for j = 0 to n-1 do
      let b1 = bul1 i j n in
      let b2 = bul2 i j n in
      for k = 1 to mat_P.(i).(j)/2 do (* 0 tour si mat_P.(i).(j) < 0 *)
        u := b1 :: b2 :: !u
      done;
    done;
  done;
  !u;;
```

Question 9 – Les fonctions `bul1` et `bul2` s'exécutent en temps $\mathcal{O}(n)$.

Dans la fonction `mcgarvey`, la boucle `for i` exécute n tours ; pour chacun de ces tours la boucle `for j` effectue n tours de boucle ; pour chacun de ces tours, on a un appel à la fonction `bul1` et un appel à la fonction `bul2` (soit un temps $\mathcal{O}(n)$) plus $\mathcal{O}(q)$ tours pour la boucle `for k`.

Finalement, le temps d'exécution est en $\mathcal{O}(n^3 + n^2 q)$

Question 10 – Si on note i le numéro d'un vainqueur de Condorcet, alors pour tout autre candidat j, il y a plus (ou autant) de personnes qui préfèrent i que j. En d'autres termes, le candidat i gagnerait ou serait ex aequo dans n'importe quel duel l'opposant un autre candidat. Il est donc naturel de qualifier i de "vainqueur".

Question 11 –

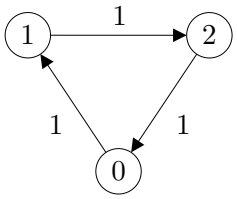
```
let est_vainq (mat_P: int array array) (i: int) =
  let n = Array.length mat_P in
  let j = ref 0 in
  while !j < n && mat_P.(i).(!j) >= 0 do
    incr j
  done;
  !j = n;;
```

```
let condorcet (mat_P: int array array): candidat list =
  let n = Array.length mat_P in

  let rec aux = function
    | i when i = n -> []
    | i when est_vainq mat_P i -> i :: aux (i+1)
    | i -> aux (i+1)
  in

  aux 0;;
```

Question 12 – On peut par exemple considérer l'urne :

```
[[0;1;2]; [1;2;0]; [2;0;1]]
```

Le graphe correspondant est :



Il n'y a donc pas de vainqueur de Condorcet.

Question 13 –

Remarque. Il me semble que la terminologie "boucle" est en général utiliser pour faire référence à un arc d'un sommet à lui-même. Ce que l'énoncé appelle "boucle" s'appelle plutôt un "circuit" ou un "cycle". Dans la suite, on utilisera "boucle" comme l'énoncé.

Soient i et j des candidats. On note $I'[i,j]$ le maximum des poids des chemins sans boucle de i à j. Il s'agit de montrer que $I[i,j] = I'[i,j]$. Étant donné que tout chemin sans boucle est un chemin, on a $I[i,j] \geqslant I'[i,j]$. Il suffit donc de montrer que $I[i,j] \leqslant I'[i,j]$. Dans la suite on note $\omega(C)$ le poids d'un chemin C.

Soit C_1 le chemin $i = i_0 \to i_1 \to \ldots \to i_k = j$ de poids maximum parmi les chemins allant de i à j. On a donc $\omega(C_1) = I[i,j]$. À partir de C_1, on peut extraire un chemin C_2 sans boucle ne passant que par des arcs de C_1. Pour cela :

- Si C_1 est sans boucle, alors $C_2 = C_1$ convient.
- Sinon, il existe $\ell_1 < \ell_2$ tels que $i_{\ell_1} = i_{\ell_2}$. Soit C_1' le chemin égal à C_1 auquel on a supprimé la boucle entre i_{ℓ_1} et i_{ℓ_2} :

$$i_0 \to i_1 \to \ldots \to i_{\ell_1 - 1} \to i_{\ell_1} = i_{\ell_2} \to i_{\ell_2 + 1} \to \ldots \to i_{\ell_2}$$

  Alors C_1' ne passe que par des arcs de C_1 et est de longueur strictement inférieure.

On itère le deuxième point ci-dessus tant que le chemin comporte une boucle. À la fin, on obtient un chemin C_2 sans boucle qui ne passe que par des arcs de C_1. Comme C_1 est de poids maximum parmi tous les chemins de i à j, on a $\omega(C_1) \geqslant \omega(C_2)$. Par définition de ω, comme toutes les arcs de C_2 sont dans C_1, on a $\omega(C_1) \leqslant \omega(C_2)$ et donc $I[i,j] = \omega(C_1) = \omega(C_2)$. Comme C_2 est un chemin sans boucle, on peut conclure :

$$I[i,j] = \omega(C_2) \leqslant I'[i,j].$$

Question 14 – Soient i, j, k des sommets distincts. On note C_1 un chemin de poids maximum entre i et j, soit C_2 un chemin de poids maximum entre j et k, et soit C le chemin entre i et k défini comme la concaténation de C_1 et C_2. Alors :

$$\omega(C_1) = I[i,j], \qquad \omega(C_2) = I[j,k], \qquad \omega(C) \leqslant I[i,k], \qquad \omega(C) = \min(\omega(C_1), \omega(C_2)).$$

En combinant ces égalités et inégalités :

$$\min(I[i,j], I[j,k]) \leqslant I[i,k].$$

Question 15 –

Remarque. L'algorithme de Floyd-Wharshall n'est plus au programme depuis la réforme de 2021.

Pour $i \neq j$ deux numéros de candidats et $k \in [\![0; n]\!]$, on note $C_k[i,j]$ l'ensemble des chemins sans boucle entre i et j dont les sommets intermédiaires sont tous dans $[\![0; k-1]\!]$. On note aussi $I_k[i,j]$ le poids maximum parmi les chemins de $C_k[i,j]$ (on a donc $I[i,j] = I_n[i,j]$). Soient $i \neq j$ deux numéros de candidats et $k \in [\![0; n-1]\!]$, essayons de calculer les $I_{k+1}[i,j]$ en fonction des $I_k[i',j']$:

(1^{er} cas) Si le poids $I_{k+1}[i,j]$ est atteint par un chemin de $C_k[i,j]$ (c'est à dire que le sommet k n'est pas un sommet intermédiaire du chemin). Alors $I_{k+1}[i,j] = I_k[i,j]$.

($2^{\text{ème}}$ cas) Sinon, le poids $I_{k+1}[i,j]$ est atteint par un chemin C passant par le sommet intermédiaire k. Puisque C est sans boucle, on a $k \notin \{i,j\}$. Le chemin C peut alors être décomposé comme la concaténation de deux chemins C_1, C_2 où $C_1 \in C_k[i,k]$ et $C_2 \in C_k[k,j]$. Le poids de C vérifie $\omega(C) = \min(\omega(C_1), \omega(C_2))$ et on a donc $I_{k+1}[i,j] = \min(I_k[i,k], I_k[k,j])$.

En combinant ces résultats, pour tout $i \neq j$ et $k \in [\![0, n-1]\!]$, si on note $m_{i,j}$ le coefficient dans la matrice d'adjacence du graphe de préférence, alors :

$$\begin{cases} I_0[i,j] = m_{i,j} & \\ I_{k+1}[i,j] = \max\Big(I_k[i,j], \min(I_k[i,k], I_k[k,j])\Big) & \text{Si } i \neq k \text{ et } j \neq k \\ I_{k+1}[i,j] = I_k[i,j] & \text{Si } i = k \text{ ou } j = k \end{cases}$$

Une autre manière de l'exprimer, serait d'autoriser $i = j$ et de poser $I_k[i,j] = +\infty$ dans ce cas :

$$\begin{cases} I_0[i,j] = m_{i,j} & \text{Si } i \neq j \\ I_0[i,j] = +\infty & \text{Si } i = j \\ I_{k+1}[i,j] = \max\Big(I_k[i,j], \min(I_k[i,k], I_k[k,j])\Big) & \end{cases}$$

```
(* ATTENTION: utiliser (let mat2 = Array.copy mat) ne permet pas de
   copier correctement une matrice: les sous-tableaux de mat et de
   mat2 seraient les mêmes, donc lorsqu'on modifierait mat cela
   modifierait aussi mat2 *)
let copy_mat (mat: 'a array array): 'a array array =
  let n = Array.length mat in
  let mat2 = Array.make n [||] in
  for i = 0 to n-1 do
    mat2.(i) <- Array.copy mat.(i)
  done;
  mat2;;
```

```
let intermediaire (mat_P: int array array): int array array =
  let n = Array.length mat_P in
  let mat_I_k = ref (copy_mat mat_P) in
  for k = 0 to n-1 do
    (* print_mat !mat_I_k; *)
    let mat_I_kp1 = Array.make_matrix n n 0 in
    for i = 0 to n-1 do
      for j = 0 to n-1 do
        if i = j || i = k || j = k then
          mat_I_kp1.(i).(j) <- !mat_I_k.(i).(j)
        else
          mat_I_kp1.(i).(j) <- max !mat_I_k.(i).(j)
                                   (min !mat_I_k.(i).(k) !mat_I_k.(k).(j));
      done;
    done;
    mat_I_k := mat_I_kp1;
  done;
  !mat_I_k;;
```

Question 16 – Chaque tour de la boucle `(for k)` s'exécute en temps $\mathcal{O}(n^2)$. Donc le temps d'exécution total est en $\mathcal{O}(n^3)$.

Question 17 – On pourrait effectivement adapter l'algorithme de Dijkstra pour calculer la matrice d'adjacence du graphe intermédiaire de Schulze. Il faudrait appliquer l'algorithme de Dijkstra n fois, chaque appel donnant une ligne de la matrice d'adjacence. Pour rappel, étant donné un graphe connexe G et un sommet i, l'algorithme de Dijkstra habituel permet de construire un arbre couvrant T de G tel que pour tout sommet j, l'unique chemin de i à j dans T est de **longueur minimal** parmi les chemin de i à j dans T. Dans la version modifiée de l'algorithme de Dijsktra présentée ici, on construit un arbre couvrant T' de G tel que pour tout sommet j, l'unique chemin de i à j dans T' est de **poids maximal** parmi les chemin de i à j dans T.

Soit i un sommet, on calcule tous les $I[i,j]$ de la manière suivante :

- Soit $A = [\![0, n-1]\!] \setminus \{i\}$ l'ensemble des sommets j pour lesquels on n'a pas encore déterminé $I[i,j]$.
- Tant que A est non vide :
  - Pour chaque $j \in A$, on définit :

$$p_j = \max\left(\left\{\min\left(I[i,k], m_{k,j}\right) : k \notin A, k \neq i\right\} \cup \left\{m_{i,j}\right\}\right)$$

    p_j est donc le poids maximal d'un chemin entre i et j dont tous les sommets sauf le dernier (qui est j) ne sont pas dans A.
  - On choisit $j \in A$ le sommet pour lequel p_j est maximal. On a alors $I[i,j] = p_j$.
  - On supprime j de A.

Pour montrer que cet algorithme fonctionne, il faut montrer qu'à chaque étape, p_j est bien le poids maximal d'un chemin entre i et j. Supposons que ce ne soit pas le cas, plaçons nous à la première étape qui pose problème, notons j le sommet sélectionné, notons C_1 le chemin entre i et j sélectionné et notons C_2 un chemin quelconque entre i et j. Soit j' le premier sommet de C_2 qui appartient à A et soit k le sommet qui précède j' dans C_2, alors le poids de C_2 vérifie $\omega(C_2) \leqslant \min(I[i,k], m_{k,j'}) \leqslant p_{j'}$. Comme l'algorithme a sélectionné le sommet j, on a $p_{j'} \leqslant p_j = \omega(C_1)$. Finalement, $\omega(C_2) \leqslant \omega(C_1)$ pour tout C_2 et donc C_1 est bien un chemin entre i et j de poids maximum, c'est à dire que $p_j = \omega(C_1) = I[i,j]$.

Question 18 –

⋆ Pour $i \neq j$, on a $S[i,j] = I[i,j] - I[j,i] = -S[j,i]$. De plus, $S[i,i] = 0$ donc la matrice est antisymétrique.
⋆ D'après la question 3, tous les coefficients de la matrice d'adjacence ont la même parité que p. Au vu de la définition du poids d'un chemin, le poids d'un chemin est aussi de la parité de p et $I[i,j]$ est de la parité de p. En réduisant modulo 2 :

$$S[i,j] \equiv I[i,j] - I[j,i] \equiv p - p \equiv 0 \mod 2.$$

Question 19 –

```
let graphe_schulze (mat_I: int array array): int array array =
  let n = Array.length mat_I in
  let mat_S = Array.make_matrix n n 0 in
  for i = 0 to n-1 do
    for j = 0 to n-1 do
      mat_S.(i).(j) <- mat_I.(i).(j) - mat_I.(j).(i)
    done;
  done;
  mat_S;;
```

Question 20 –

```
let schulze (n: int) (u: urne): candidat list =
  let mat_P = depouillement 3 u in
  let mat_I = intermediaire mat_P in
  let mat_S = graphe_schulze mat_I in
  condorcet mat_S;;
```

Question 21 –
La fonction `depouillement` s'exécute en temps $\mathcal{O}(n^2 p)$.
La fonction `intermediaire` s'exécute en temps $\mathcal{O}(n^3)$.
La fonction `graphe_schulze` s'exécute en temps $\mathcal{O}(n^2)$.
La fonction `condorcet` s'exécute en temps $\mathcal{O}(n^2)$.
Finalement, la fonction `condorcet` s'exécute en temps $\mathcal{O}\Big(n^2(n+p)\Big)$.

Question 22 – Soient i, j, k trois candidats tels que $iR_S j$ et $jR_S k$. On a donc :

$$I[i,j] > I[j,i], \qquad\qquad I[j,k] > I[k,j].$$

Notons que i, j, k sont deux à deux distincts :

- $i \neq j$ car $S[i,i] = 0$ et donc $iR_S i$ est faux.
- $i \neq k$ car la matrice d'adjacence dans le graphe de préférence est antisymétrique et donc on ne peut pas avoir $S[i,j] > 0$ et $S[j,i] > 0$.
- De même, $j \neq k$.

Soit C_1 (resp. C_2) un chemin entre i et j (resp. j et k) de poids $I[i,j]$ (resp. $I[j,k]$) dans le graphe de préférence. Les chemins C_1 et C_2 sont donc de poids maximum. On doit montrer la relation $iR_S k$, c'est à dire que $I[i,k] > I[k,i]$.

Soit C_3 le chemin entre i et k défini comme la concaténation de C_1 et C_2 alors $\omega(C_3) \leqslant I[i,k]$. Soit C un chemin de k à i dans le graphe de préférence et supposons par l'absurde que $\omega(C) \geqslant \omega(C_3)$. On distingue les deux cas de l'indication de l'énoncé :

⋆ Si $I[i,j] \leqslant I[j,k]$, alors :

$$\omega(C_3) = \min(\omega(C_1), \omega(C_2)) = \omega(C_1) = I[i,j]$$

Soit C_4 le chemin de j à i défini comme la concaténation de C_2 et C. Alors :

$$\omega(C_4) = \min(\omega(C_2), \omega(C))$$

Or, $\omega(C) \geqslant \omega(C_3) = \omega(C_1)$, et $\omega(C_2) = I[j,k] \geqslant I[i,j] = \omega(C_1)$. On en déduit que :

$$\omega(C_1) \leqslant \min(\omega(C_2), \omega(C)) = \omega(C_4).$$

Finalement, comme C_4 est un chemin de j à i :

$$I[i,j] = \omega(C_1) \leqslant \omega(C_4) \leqslant I[j,i].$$

Contradiction avec l'hypothèse initiale iR_Sj.

⋆ Le cas $I[i,j] > I[j,k]$, se traite de la même façon avec C_4 le chemin entre k et j défini comme la concaténation de C et C_1.

Question 23 – On construit un chemin $i_1 \to \ldots \to i_k$ dans le graphe de préférence de la manière suivante :

- i_1 est un sommet quelconque
- Supposons que i_ℓ ait été construit.
  - Si tous les arcs issus de i_ℓ sont de poids strictement positif alors le chemin est terminé (c'est à dire que $k = \ell$).
  - Sinon, on définit $i_{\ell+1}$ comme l'un des sommets tel que $S[i_\ell, i_{\ell+1}] \leqslant 0$. Si $i_{\ell+1}$ appartenait déjà au chemin, il est terminé (c'est à dire que $k = \ell + 1$).

Notons que cette procédure s'arrête puisque tous les sommets du chemin (sauf éventuellement le dernier) sont distincts et que le nombre de sommets du graphe est fini. Par construction, on a $i_\ell R_S i_{\ell-1}$ pour tout ℓ.

Supposons par l'absurde que la procédure s'arrête car un sommet appartenait déjà au chemin. On a alors $i_k = i_{k'}$ pour un certain $k' < k$ et donc la question précédente indique que $i_k R_S i_k$ ce qui est absurde car $S[i_k, i_k] = 0$.

Ainsi, la procédure s'arrête nécessairement pour un sommet i_k tel que tous les arcs issus de i_k sont de poids strictement positif. Donc pour tout $i \neq i_k$: $S[i_k, i] > 0$ et i_k est un vainqueur de Schulze.

Question 24 –

⋆ Pour fixer les idées, les variables propositionnelles sont notées $x_1, x_2, x_3 \ldots, x_n$, et on suppose que la formule φ est représentée par un tableau de listes d'entiers dans lequel les littéraux x_i et $\neg x_i$ correspondent aux entiers i et $-i$. Une interprétation est donnée par un tableau de booléens de taille $(n+1)$ dont la case d'indice i est la valeur de vérité de x_i (la case d'indice 0 a une valeur quelconque, disons `false`).

⋆ Étant donnée une interprétation, on peut vérifier en temps $\mathcal{O}(m)$ si φ est vraie. En effet, une clause est vraie lorsqu'au moins un de ses littéraux est vrai (il suffit de parcourir la liste pour le vérifier). La formule φ est vraie lorsque toutes ses clauses sont vraies (il suffit de parcourir chaque liste du tableau pour le vérifier).

⋆ Pour vérifier si φ est satisfiable, on peut générer exhaustivement toutes les interprétations et tester si l'une d'elles rend φ vraie. Pour générer toutes les interprétations, on peut utiliser une fonction récursive :

```
let gen_exhaustive (n: int) =
  let interp = Array.make (n+1) false in
  let rec aux = function
    | 0 -> (* On teste si interp rend phi vraie *)
    | i -> aux (i-1);
           interp.(i) <- true;
           aux (i-1);
           interp.(i) <- false;
  in
  aux n;;
```

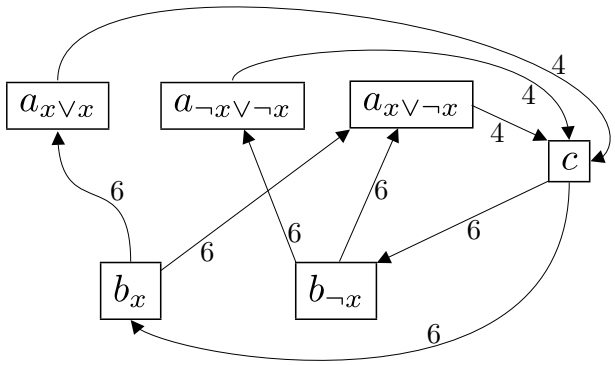
Lors d'un appel à `gen_exhaustive`, l'arbre des appels récursifs est un arbre binaire complet à 2^n feuilles, son nombre de noeuds est donc en $\mathcal{O}(2^n)$. De plus, pour chacune de ces feuilles, on teste si une interprétation rend φ vraie en temps $\mathcal{O}(m)$.

Le temps d'exécution total est donc en $\mathcal{O}(m2^n) + \mathcal{O}(2^n) = \mathcal{O}(m2^n)$.

Question 25 – Soit $\varphi = (x \vee x) \wedge (\neg x \vee \neg x)$. Après la première boucle, on a :

$$\psi = (x \vee x) \wedge (\neg x \vee \neg x) \wedge (x \vee \neg x)$$

On obtient le graphe de préférence suivant dans lequel on a tracé uniquement les arcs de poids strictement positif :



Il n'est pas possible de choisir $k = 1$ candidat parmi x et $\neg x$ de sorte que c gagne. En effet :

- Si on choisit x, alors on ne choisit pas $\neg x$ et il n'existe pas de chemin de poids strictement positif entre c et $a_{\neg x\vee\neg x}$. Ainsi : $I[c, a_{\neg x\vee\neg x}] < 0 < 4 = I[a_{\neg x\vee\neg x}, c]$ et donc c ne gagne pas.
- De même, si on choisit $\neg x$ alors c ne gagne pas car $I[c, a_{x\vee x}] < I[a_{x\vee x}, c]$.

Question 26 – Soit x une variable propositionnelle.

$\star$ Le graphe de préférence G contient les trois sommets $a_{x\vee\neg x}$, b_x et $b_{\neg x}$. Pour que le champion c gagne, il faut en particulier que $I[c, a_{x\vee\neg x}] > I[a_{x\vee\neg x}, c]$. Par construction du graphe, le seul arc sortant de $a_{x\vee\neg x}$ de poids strictement positif a pour autre extrémité c et donc $I[a_{x\vee\neg x}, c] = 4$ en utilisant la question 13. Dans G, il y a deux chemins de c à $a_{x\vee\neg x}$ de poids strictement positif :

$$c \to b_x \to a_{x\vee\neg x} \qquad \text{et} \qquad a \to b_{\neg x} \to a_{x\vee\neg x}.$$

Ainsi pour avoir $I[c, a_{x\vee\neg x}] > 4$, il faut que l'un de ces deux chemins soit accessible, c'est à dire qu'on doit avoir choisi x ou $\neg x$.

$\star$ Comme k est le nombre de variables propositionnelles, on ne peut pas choisir x et $\neg x$. Sinon, il existerait au moins une autre variable y telle que y et $\neg y$ ne seraient pas choisis, ce qui contredirait le point précédent.

Question 27 – On le montre par double implication.

($\Rightarrow$) On suppose que φ est satisfiable et on doit montrer que l'instance de CONTROL-ADD-ALT donne une réponse positive. Soit f une interprétation qui rend vraie φ. On choisit les k candidats de la manière suivante : pour chaque variable propositionnelle x :

- Si x est vraie pour f, alors on choisit x.
- Sinon, on choisit $\neg x$.

On doit montrer que c est un vainqueur de Schulze dans la graphe obtenu noté G' :

- On remarque d'abord que pour tout littéral ℓ de G', on a $I[c, b_\ell] = 6$ et $I[b_\ell, c] \leqslant 4$ car tous les arcs entrants de c sont de poids 4 ou négatif. Ainsi, $S[c, b_\ell] > 0$.
- Soit cl une clause de la forme $x \vee \neg x$ où x est une variable propositionnelle. Alors $I[a_{cl}, c] = 4$ et $I[c, a_{cl}] = 6$ car l'un des deux chemins suivants appartient à G' :

$$c \to b_x \to a_{x \vee \neg x} \qquad \text{ou} \qquad a \to b_{\neg x} \to a_{x \vee \neg x}.$$

- Soit cl une clause de φ. Comme f rend φ vraie, elle rend cl vraie. Par construction de G', on a choisi au moins un littéral ℓ contenu dans cl. Ainsi, $I[cl, c] = 4$ et $I[c, cl] = 6$ car le chemin suivant appartient à G' :

$$c \to b_\ell \to a_{cl}$$

Finalement, pour tout sommet $c' \neq c$ de G', on a $S[c, c'] > 0$ et donc c gagne.

($\Leftarrow$) On suppose que l'instance de CONTROL-ADD-ALT donne une réponse positive et on doit montrer que φ est satisfiable. Soit G' le graphe tel que c gagne. On définit une interprétation f de la manière suivante : pour chaque variable propositionnelle x :

- Si x est choisie dans G', alors x est vraie pour f.
- Si $\neg x$ est choisie dans G', alors x est fausse pour f.

Notons que la question précédente assure qu'un et un seul des deux points ci-dessus est vérifié. Montrons que f rend vraie φ. Soit cl une clause de φ. Comme c gagne dans G', on a $S[c, a_{cl}] > 0$. Or $I[a_{cl}, c] = 4$ et donc on a nécessairement $I[c, a_{cl}] = 6$, ce qui signifie qu'il existe un chemin de poids 6 dans G' entre c et a_{cl}. Il existe donc un littéral de cl qui a été choisi dans G'. Ainsi ce littéral est vrai pour f et donc cl est vraie pour f. Finalement, φ est vraie pour f.

Question 28 – Par contraposée, on suppose qu'il existe un algorithme A en temps polynomial en le nombre de candidats qui résout CONTROL-ADD-ALT et on montre que la conjecture est fausse.

Soit φ une instance de SAT avec m littéraux et I l'instance de CONTROL-ADD-ALT obtenue par l'algorithme de l'énoncé. Le nombre de candidats dans l'instance I est polynomiale en m et la construction de I s'exécute en en temps polynomial en m. On résout ensuite I avec l'algorithme A en temps polynomial en m (une composition de polynômes est un polynôme). Avec la question précédente, on sait donc si φ est satisfiable ou non. La procédure complète s'exécute en temps polynomial en m et la conjecture et donc fausse.